**Remarks**

**Remarks Regarding Amendments to the Specification**

1. In paragraph [0006] on page 1, the reference to canceled claim 11 is replaced by a reference to claim 14 to restore proper form.

2. It is requested that the formatting of paragraph [0004] be changed to a section heading.

**Remarks Regarding Amendments to the Claims**

3. Claim 11 is canceled and re-introduced as new claim 14 to restore proper form after being objected to in the Final Office Action.

**Remarks Regarding Claim Rejections under 35 U.S.C. 103(a)**

4. The Final Office Action rejects claims 1, 3-12 under 35 U.S.C. 103(a) as being unpatentable over Thomsen et al. (USPN 5,987,246) in view of Simonyi (USPN 5,790,863). Applicant traverses the rejections. Since claim 11 now appears as claim 14, claims 1, 3-10, 12-14 remain pending.

5. A proper rejection under 35 U.S.C. 103(a) requires that the prior art references must teach or suggest all the claim limitations. See MPEP 706.02(j). Thomsen et al. in view of Simonyi does not disclose or suggest at least one feature recited in claims 1, 3-10, 12-14.

6. Independent claim 1 is directed to a method which combines

    a) modeling a hierarchical data structure by input means with

    b) dynamically computing parts of such data structure through functional expressions employing the use of dynamic binding.

    Thomsen et al. in view of Simonyi neither discloses nor suggests the feature of an "expression being able to contain a reference using dynamic binding to refer to at least one other element" as recited in claim 1.

An expression according to Thomsen could only contain references to other elements being statically bound (hard-wired) by connecting function inputs to outputs through graphical user interactions (see Thomsen, 2:47 – 50). Such references (inputs) cannot change (connect to a different output element) once execution begins (at "run time"). As a result, Thomsen provides no mechanism for functional expressions to automatically respond to the insertion or deletion of elements (at "run time").

For an example, given an order entry system as described in Fig. 3, consider the possible addition of an "Item 4" or a deletion of "Item 3". Based on Thomsen's disclosure, it would be impossible to construct an expression as depicted in Fig. 3, 11, which could compute the total sum of amounts from a set of order items where that set of order items would grow or shrink dynamically as new items are added to or deleted from the hierarchy, without requiring the user to change such expression on each occurrence.

Simonyi does not remedy that deficiency.

Unlike Thomsen in view of Simonyi, the preferred embodiment with the features of claim 1 expressly enables expressions to automatically respond to the insertion or deletion of elements (at "run time"). See 1:[0014], 1:[0016], 3:[0055] – [0056].

The Final Office Action does not address the above feature of claim 1. However, had it been considered, there would clearly be no motivation in view of Thomsen et al. combined with Simonyi to include such a feature.

In both Thomsen and Simonyi, the subject of modeling is not a dynamic data structure as recited in claim 1, but a program structure. As the latter is typically static by nature (does not change at "run time"), and references would point to elements of that static program structure, the targets of said references do not change dynamically, so there is no motivation to contemplate using dynamic binding in such references. Instead, when compared to conventional static binding in that context, the difficulties of implementing dynamic binding and performance considerations would advise against the use of dynamic binding.

Remarkably, the preferred embodiment of claim 1 avoids the hierarchical modeling of its "program fragments" (functional expressions), but represents those functional expressions in text form instead (see, Fig. 3, 11). Thus, data flow is not represented graphically, but implied in references of those text-based functional expressions. Kodosky et al. (USPN 4,901,221), incorporated in its entirety by Thomsen et al., explicitly teaches away from such a concept stating:

> Current attempts to develop text based flow software systems generally have been unsuccessful. This lack of success may be due to difficulties in perceiving parallels between text based software and actual data flow, and it may be due to the fact that the text form of data flow generally suffered from the same difficulties in representing conditional and iterative functions as did the diagram form of data flow. (Kodosky et al., 3:30 – 37)

Likewise, Simonyi teaches replacing text-based expressions with a graphical representation.

In summary, as the disclosures of both Thomsen and Simonyi are directed at modeling a rather static program structure instead of a dynamic data structure, there is no motivation in their disclosures for addressing structural dynamics, which claim 1 does through its feature of a "reference using dynamic binding".

For at least the foregoing reasons, applicant submits that claim 1 is neither anticipated nor suggested by Thomsen in view of Simonyi.

7. Claims 3-10 and 12-14 depend from claim 1. Therefore, these claims are neither anticipated nor suggested by Thomsen in view of Simonyi for at least the reasons given above with respect to claim 1. Moreover, these claims recite additional features not disclosed or suggested by Thomsen in view of Simonyi.

8. Regarding claim 3, Thomsen does not disclose the feature of modeling "in an object-oriented way, by having a class structure represent the configuration and properties of a number of aggregate structures of the same kind separately from their individual contents".

While Thomsen briefly mentions "Alternatively, the graphical program is an object-oriented program that is not a data flow program, as desired" (6:34 – 36), he does not disclose steps to model such a program. It appears that the term object-oriented refers to program code outside the scope of Thomsen's modeling method. In any case, Thomsen does not disclose anything related to the above data modeling features recited in claim 3.

Simonyi does not remedy that deficiency. Like Thomsen, Simonyi discloses modeling a program structure and does not mention object-orientation at all. However, applying object-oriented modeling according to claim 3 to a program tree would mean creating classes for "aggregate structures of the same kind", which would have to be non-leaf nodes of the program tree. Since such nodes are operators, one would, for example, create a common class for all occurrences of the "+" operator having two operands. Obviously, doing so would make no sense to one skilled in the art.

Unlike with object-oriented data modeling according to claim 3, which would enable the uniform treatment of data structures such as order items (see 4:[0063], Fig. 6), there would be no motivation to apply modeling with the features recited in claim 3 to a program structure as disclosed by Thomsen in view of Simonyi.

Further, generally available knowledge with regard to object-oriented software construction teaches against combining functional expressions recited in claim 1 with object-oriented data modeling recited in claim 3, as doing so would violate the principle of encapsulation, a form of information hiding which limits direct access to an object's data elements to member functions of that object. In contrast, the preferred implementation of claim 3 makes extensive use of dynamic references across object boundaries (see 4:[00072] with a "COMPONENTS(Class)" clause referring to subordinate elements across one or more object boundaries).

9. Regarding claim 4, Thomsen (4:25 – 35) in view of Simonyi does not disclose or suggest a "parameter for an element's representation mode" or a "parameter for an element's editing mode". Such parameters would have to be differentiated

from an element (node) itself and the element's contents (input and output variables). The Final Office Action does not logically explain which parts of Thomsen's disclosure relate to the recited features of claim 4.

10. Regarding claim 5, Thomsen (2:45 – 55) in view of Simonyi does not disclose or suggest parameters according to claim 4, or "multiple sets of parameters", let alone "one of said parameters sets becoming effectual depending on the results of manipulatable expressions". The Final Office Action does not logically explain which parts of Thomsen's disclosure relate to the recited features of claim 5.

11. Regarding claim 6, while Thomsen (5:15 – 20) discloses acquiring data from an instrument and storing such data, Thomsen in view of Simonyi does not disclose or suggest the feature of storing "all information, including meta information" or the feature of storing such information persistently (as opposed to temporarily in volatile memory).

12. Regarding claim 7, while Thomsen (10:7 – 12) discloses receiving "invalid input due to a user error", Thomsen in view of Simonyi does not disclose or suggest the feature of "marking the result of an expression invalid if, and only if the expression was modified or the contents of an element referenced by the expression were modified or became invalid". Thomsen discloses how permanent user errors are dealt with, which would preclude an expression of producing a result in the first place, while the above feature of claim 7 is directed to a temporary state of invalidity, as legitimate changes of the expression and/or its input data cause the expression's result to be temporarily out of date until recomputed.

    In addition, Thomsen in view of Simonyi does not disclose or suggest the feature of "updating the result on an expression not until it is needed for representation or in the course of computing another result", which in conjunction with the invalidity mark mentioned above constitutes a lazy evaluation technique. While Thomsen explicitly states that "[w]hen the program is executed, each node executes when it has received data at all of its inputs" (1:63 – 65), the lazy evaluation feature as recited in claim 7 defers execution of an expression until its results are needed.